AccountVault

Dan Blossom
Marist College
Professor Pablo Rivas
MSCS 630 Security Algorithms and Protocols
April 14th, 2019

Abstract: This paper will describe an Android application that will store not only a users password and username to a specific account, but other details they might need at a later date such as security questions. In order to protect the users privacy the various items (email, password, etc) will be encrypted using the Advanced Encryption Standard (AES) and stored in a SQLite database.

Introduction: This project is being created to learn if it's a feasible solution to use AES encryption to store a users account information into a SQLite database. The project is similar to any password manager but instead of storing account passwords for later retrieval it'll store other account information. This information could be usernames, email addresses, passwords, and security questions. The thought is to take the information, encrypt it, and store it. When the user needs the information it'll be decrypted and displayed for them.

Background: Whether it's a well known application like LastPass or a random app from a mobile store, there are no shortage of password manager applications. An application I haven't seen is one that will store other account information that is often required for verifying identity. This information could be email addresses or security questions. Since I do not answer security questions with the real answer and have multiple email addresses it's difficult to keep track of it

all. So, my thought for this project is to see if taking a password manager to the next level and store all account information is a security safe solution.

Methodology: The methodology I am currently working on is that a user can enter information about an account they own (bank, social media, online shopping, etc) and it will be encrypted before being stored into a SQLite database. The encryption key will be randomly generated and used in encrypting the plain text input. Both the encrypted text and randomly generated key will be stored in the database. When the user selects the account to retrieve the encrypted key will be used to decrypt the hex encrypted string to the original plain text.

Experiments: Passing a 128-bit plain text (in hex) string along with a 128-bit hex key to an AES algorithm created for previous educational purposes can be a bit limiting. 1) The plain text string has to be in hex and 2) the plain text hex string cannot be longer than 128-bits. I decided to combine both 1 & 2 mentioned above into one experiment. I added a method that converts a plain text string of any length to 128-bit hex chunks to pass into original AES algorithm. Once those tests were successful, next was to implement an algorithm to decrypt the encrypted text which proved to be less trivial. However, through research, I implemented the algorithms to decrypt the encrypted text given the same key. A few methods were added to perform the decryption on the encrypted text as well as a few new lookup tables. Most importantly for inversing the Mix Columns, which simplified trying to use Galois Multiplication to calculate the Inverse of the original Mix Columns.

Here are two tests:

```
$ java Driver < test2.txt
        Original: hereisastringlongerthan16charsithink
```

Encrypted:

87E7D255FFCBB81BA269500914CCCA1C33FC048AE140C975FF847F

924F7A506FF9079715D75E9A1CA28333DF55903CB7

Decrypted: hereisastringlongerthan16charsithink

$ java Driver < test3.txt

Original: yay we can encrypt and decrypt long text!!

Encrypted:

53CF80DA1C3712BA4019B8C39F2A53A907B86295BF9347067EB8341

91376186F3E9282C50F5B87394F1D191A8BC8FFF7

Decrypted: yay we can encrypt and decrypt long text!!

Discussion: The experiments performed prove that I'll be able to take any string of text, encrypt it and using the same key, decrypt it. This proves to be an effective way to store sensitive information into a SQLite database. However, without multiple layers of security, this application will not be completely secure. There would need to be a password on the application itself as well as encryption on the database. Also, storing the key used to encrypt doesn't seem security safe either. This project is just focusing on using AES to encrypt text before storing it.

Conclusion: Since AES hasn't been broken it'll prove to be an effective way to encrypt sensitive information. In this paper I've proved we can incorporate an existing AES algorithm to securely encrypt and decrypt long strings of text. The next stage will be to design the application using Android Studio and incorporate the AES encryption and decryption methods used for this paper into that application.

References: I do not have any scholarly references used for this paper or these experiments, but the below websites were extremely helpful in implementing the biggest hurdle of the decryption steps, the inverse of mix columns.

https://ipfs.io/ipfs/QmXoypizjW3WknFiJnKLwHCnL72vedxjQkDDP1mXWo6uco/wiki/Rijndael_mix_columns.html

https://crypto.stackexchange.com/questions/2569/how-does-one-implement-the-inverse-of-aes-mixcolumns

https://en.wikipedia.org/wiki/Rijndael_MixColumns